Text-Representation Technique

I have chosen the word embedding for text representation because it is powerful when we are dealing with a large dataset. The bag of word is better when we are working with a small dataset. From my recent researches and my investigations, I found that the word embedding is capable of giving me a good result and is working well for Natural Language Processing model. As we know, Keras Library offers embedding layers that can be used to train the neural network.
My idea is to use the convolutional neural network as machine learning approach. In my case, the word embedding is a perfect match.
I will be training my word embeddings during the training of the neural network by adding an additional layer. As I said before, this is already set and ready by Keras Library. What I need to do is mapping my dataset from sequences of words to sequences of word integers. The reason why it is important to the map is because it is not possible to multiply words by weight. I don't think there is a concept for multiplying words, each word needs to have its own unique number; then, I can use an embedding layer to map each word integer to a corresponding word of vector.
I was thinking of using the one-hot encoding for the text processing. After my research on the web, it mention that it is working fine but it will take too much space so it is not a perfect idea, also the one-hot encoding is not geometrically useful because each word vector will be equal distance, for example (car is just close to building). As we know machine learning is just a geometry problem, so we need a better way to convert the word to vector.

Machine Learning Approach

As we know, convolution neural is the most popular artificial neural network being used for analysing images. We know that it has some type of specifications being able to detect patterns. As we saw in the previous units, it works really well with images classification; we know how images were presented to in an array of pixels, the same logic applies for the text. As I mentioned previously, we just need to map each word to a specific vector space; after that, we will let the convolution neural network process that.

In my case, my dataset is on a csv file. I had to transform the text from sentences to sequences of integers using the embedding approaches.
The word embedding will give me exactly what I need a 1-dimension convolution layers.

As I have said previously, Keras Library gives us a big help to define a convolution neural network architecture to train and evaluate a CNN model. Also, I have been modifying my CNN function to get better accuracy and loss value.

```
D = 50
i = Input(shape=(T,))
x = Embedding(V, D)(i)
x = Conv1D(32, 3, activation='relu')(x)
x = MaxPool1D(3)(x)
x = Conv1D(64, 3, activation='relu')(x)
x = MaxPool1D(3)(x)
x = Conv1D(128, 3, activation='relu')(x)
x = GlobalMaxPool1D()(x)
x = Dense(1, activation='sigmoid')(x)
model = Model(i, x)
```

```
model.compile(optimizer = 'sgd', loss = 'binary_crossentropy', metrics = ['accuracy'])
model.summary()
print('Training model**')
Training = model.fit(X_train, y_train, epochs = 20, validation_data=(X_Val, y_Val))
```

After the update

```
[ ]  D = 50
     i = Input(shape=(T,))
     x = Embedding(V, D)(i)
     x = Conv1D(32, 3, activation='relu')(x)
     x = GlobalMaxPool1D()(x)
     x = Dense(10, activation='relu')(x)
     x = Dense(1, activation='sigmoid')(x)
     model = Model(i, x)
```

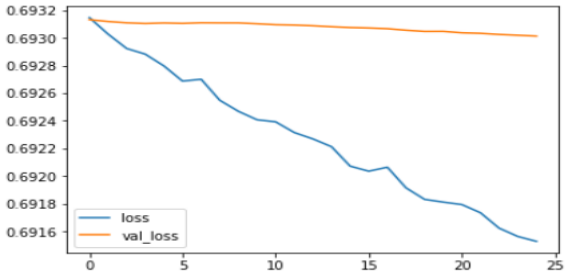
```
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
print('Training model**')
Training = model.fit(data_train, y_train, epochs = 10, validation_data=(data_val, y_Val))
model.summary()
```

As we see on the screenshot that I have used 3 convolution layers on the first capture and each one has a rectifier function (ReLU).

The 32 is how many features maps applied on the first convolution layer. I use the ReLu activation function as a general rule if we have not reached the output layer, we will stick with the rectifier function. You have noticed that every convolution layer is following with Maxpooling layer. We use it to down-sampling the input, meaning that it took the maximum value over the time dimension.

After that I add 2 convolution layers, as you see the number of features maps is doubling, after I finalised by GlobalMaxPoolin1D layer. Then, I set up a fully connection layer that has 1 hidden neuron for the activation function. At this point, I pick the sigmoid function instead of the softmax. The reason behind this choice is because I'm working with sentiment analysis task of natural language processing and my objective is to predict whether a piece of text is a positive or negative sentiment. This kind is a type of binary classification, so sigmoid is a valid choice for this type of prediction. To compile the convolution neural network, I have chosen the stochastic gradient descent as optimizer and the loss function is binary_crossentropy, I start with 25 as the number of epochs.

```
Training Accuracy: 0.6100
Validation Accuracy:  0.5450
```



The result was not good as you see in the image captured, the loss value in the validation data gets high on each epoch, this gives us an idea that model not doing well in both the training and the validation, something is wrong.
I started by minimizing the numbers of the convolution layers. Also after my research, I found out that Adam is a replacement optimization algorithm for stochastic gradient descent for training on deep learning models. So, I changed the optimizer from stochastic gradient descent to Adam optimizer. Also, I did try to cut down the numbers of epochs and I observed that the model is getting better, I modified the numbers of epochs to 10. After I ran the script, the new results looked much better than before. Validation Accuracy : 0.7300 and Training Accuracy : 1.0000, we see in the training summary that the value of the loss went down for both training and validation, that is pretty good.

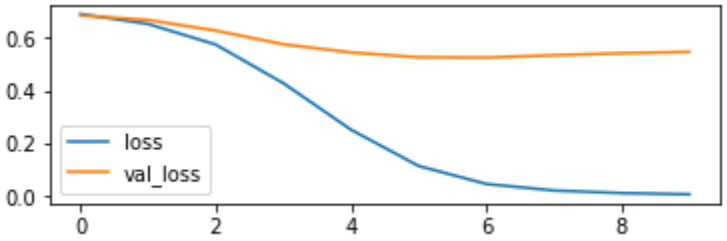
```
Training Accuracy:    1.0000
Validation Accuracy:  0.7300
```

Evaluation

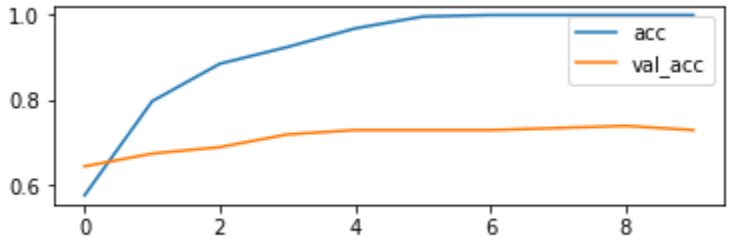
```
Training model**
Epoch 1/10
44/44 [==============================] - 7s 149ms/step - loss: 0.6896 - accuracy: 0.5771 - val_loss: 0.6841 - val_accuracy: 0.6450
Epoch 2/10
44/44 [==============================] - 6s 144ms/step - loss: 0.6520 - accuracy: 0.7979 - val_loss: 0.6671 - val_accuracy: 0.6750
Epoch 3/10
44/44 [==============================] - 6s 143ms/step - loss: 0.5741 - accuracy: 0.8857 - val_loss: 0.6270 - val_accuracy: 0.6900
Epoch 4/10
44/44 [==============================] - 6s 145ms/step - loss: 0.4283 - accuracy: 0.9250 - val_loss: 0.5754 - val_accuracy: 0.7200
Epoch 5/10
44/44 [==============================] - 6s 145ms/step - loss: 0.2527 - accuracy: 0.9686 - val_loss: 0.5445 - val_accuracy: 0.7300
Epoch 6/10
44/44 [==============================] - 6s 145ms/step - loss: 0.1159 - accuracy: 0.9964 - val_loss: 0.5265 - val_accuracy: 0.7300
Epoch 7/10
44/44 [==============================] - 6s 145ms/step - loss: 0.0481 - accuracy: 1.0000 - val_loss: 0.5254 - val_accuracy: 0.7300
Epoch 8/10
44/44 [==============================] - 6s 145ms/step - loss: 0.0234 - accuracy: 1.0000 - val_loss: 0.5342 - val_accuracy: 0.7350
Epoch 9/10
44/44 [==============================] - 6s 145ms/step - loss: 0.0137 - accuracy: 1.0000 - val_loss: 0.5416 - val_accuracy: 0.7400
Epoch 10/10
44/44 [==============================] - 6s 144ms/step - loss: 0.0090 - accuracy: 1.0000 - val_loss: 0.5467 - val_accuracy: 0.7300
Model: "model_17"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_24 (InputLayer)       [(None, 2299)]            0

 embedding_23 (Embedding)    (None, 2299, 50)          1862600

 conv1d_23 (Conv1D)          (None, 2297, 32)          4832

 global_max_pooling1d_17 (Gl  (None, 32)               0
 obalMaxPooling1D)

 dense_34 (Dense)            (None, 10)                330

 dense_35 (Dense)            (None, 1)                 11

=================================================================
Total params: 1,867,773
Trainable params: 1,867,773
Non-trainable params: 0
```

The evaluation of our testing data gives us accuracy: 81.7%. That data has never been seen before by the module. If you also see the amount of the loss value curving down, on each epoch, while the training happened, we get 100 % for the training accuracy and 73 % for the validation accuracy. The graph shows us a very good visualization on how the module is doing.

the capture gives us a detail about the loss per iteration.



We are displaying the accuracy per iteration and the result going up progressively.



Report

In this task, I'm going to explain the steps that I have used for natural language processing to predict the piece of text associated with a positive or negative sentiment. To tackle this task, I used convolution neural networks for text classification, I used Google Collab to build the model. I found it very user-friendly, and no install required other than to install the TensorFlow, but it is only one command.

```
!pip install -q tensorflow-gpu==2.7.0
```

Also, easy when it comes to import data from the csv files, I just need to upload it and put my data csv file link for each dataset.

The script that I used was divided by steps, I highlighted some of the steps. First, I started by text-preprocessing, embedding, building and training CNN and the last step is the evaluation model.

First, I declared all our imports that I needed to use. I have declared the tokenizer and pad_sequences so I can pre-process the texts and I have all the layers that I need to create 1-dimension convolution neural network.

```
[202]
      from tensorflow.keras.layers import Dense, Input, GlobalMaxPool1D
      from tensorflow.keras.layers import Conv1D, MaxPool1D, Embedding
      from tensorflow.keras.models import Model
      import pandas as pd
      from keras.models import Sequential
      from keras import layers
      from keras.preprocessing.text import Tokenizer
      from keras.preprocessing.sequence import pad_sequences
      import matplotlib.pyplot as plt
      import numpy as np
      import pandas as pd
```

```
[293] # Train Data
      data_train = pd.read_csv('/content/train.csv', names=['sentence', 'label'])
      # Validation Data
      data_val = pd.read_csv('/content/val.csv', names=['sentence', 'label'])
      # Test Data
      data_test = pd.read_csv('/content/test.csv', names=['sentence', 'label'])
```

I used the pandas to read the data that was provided to me: 3 different types of data. What is good, it was already splitted so no need to split it again (data for training, data for validation and data for testing) and rename the columns to labels and sentences.

```
#data_train.head()
#data_val.head()
data_test.head()
```

This will allow us to check the data by data_train.head.

```
# I store the reviews and the labels in  arrays

reviews = data_train['sentence'].values
label = data_train['label'].values

reviewstest = data_test['sentence'].values
labeltest = data_test['label'].values

reviewsval = data_val['sentence'].values
labelval =   data_val['label'].values
```

Here, in this command, I have stored the reviews and labels in two arrays for each dataset.

```
# Convert sentences to sequences
MAX_VOCAB_S = 20000
tokenizer = Tokenizer(num_words = MAX_VOCAB_S)
tokenizer.fit_on_texts(reviews_train)
# new variable X_train and y_train and X_test
seq_train = tokenizer.texts_to_sequences(reviews_train)
seq_val = tokenizer.texts_to_sequences(reviewsVal)
seq_test = tokenizer.texts_to_sequences(reviewstest)
```

On this part it comes to convert sequence of text into sequences of integers using the tokenizer, first I need to call fit_on_text on the trainset and we call text to sequences on trainset, validationset and testset so this will give us X_train sequences, X_Val sequences and X_test sequences.

```
[299] wordToIndex = tokenizer.word_index
      V = len(wordToIndex) + 1
      print(reviews_train[0])
      print(seq_train[0])
      V
```

At this stage, I needed to get the word index mapping and the vocabulary size V as we have proximally size 37252 tokens. Also, as we see I added an additional one because the first index starts from 1 not 0.

```
[ ]  data_train = pad_sequences(seq_train)
     print('Shape  of the data train : ', data_train.shape )
     T = data_train.shape[1]
```

```
Shape  of the data train :  (1400, 2299)
```

```
[ ]  data_val = pad_sequences(seq_val, maxlen=T)
     print('Shape  of the data val : ', data_val.shape )
```

```
Shape  of the data val :  (200, 2299)
```

```
[ ]  data_test = pad_sequences(seq_test,maxlen=T)
     print('Shape  of the data test : ', data_test.shape )
```

```
Shape  of the data test :  (400, 2299)
```

At this stage I called the pad_sequences to pad the trainset, and also o called the pad_sequences to pad the validationset using 2299 as maxlen and also I called the pad_sequences to pad my testset using the same maxlen 2299.

*****Creating of the model*****

I needed to create a model, so I will start by the embedding layer and then we will have a convolution layer followed by GlobalMaxpooling. Finally, we have two dense: the first one with rectifier function, the last dense with sigmoid activation since this is binary classification.

```
D = 50
i = Input(shape=(T,))
x = Embedding(V, D)(i)
x = Conv1D(32, 3, activation='relu')(x)
x = GlobalMaxPool1D()(x)
x = Dense(10, activation='relu')(x)
x = Dense(1, activation='sigmoid')(x)
model = Model(i, x)
```

We will fit the model for 5 training epochs with batch size of 10. After that, I called model.compile with adam optimizer, for the loss will be a binary_crossentropy and for the metrics is the accuracy.

```
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
print('Training model**')
Training = model.fit(data_train, y_train, epochs = 10, validation_data=(data_val, y_Val))
model.summary()
```

The last step is to evaluate the model on the training and validation

```
[384] loss, accuracy = model.evaluate(data_train, y_train, verbose=False)

      print("Training Accuracy: {:.4f}".format(accuracy))

      loss, accuracy = model.evaluate(data_val, y_Val, verbose=False)

      print("Validation Accuracy:  {:.4f}".format(accuracy))
```

```
Training Accuracy: 1.0000
Validation Accuracy:  0.7300
```

As we see, the accuracy for the training is **100 %** and for the validation is **73%.**

```
[387] loss, accuracy = model.evaluate(data_test, y_test, verbose=False)
      print("Testing  Accuracy:  {:.4f}".format(accuracy))
```

```
Testing  Accuracy:  0.8175
```

For the testing data accuracy, I got **81.7%** pretty good result, what is not that bad compare the accuracy that I was getting before. Based on this number, now I'm a little bit more confident than before.